Lossy Multimedia Compression (1)

Entropy-based compression as presented in the previous chapter is an important foundation for many data formats for multimedia. However, as already pointed out, it often does not achieve the compression rates required for the transmission or storage of multimedia data in many applications. Since compression beyond Entropy is not possible without losing information, that is exactly what we have to do: Lose information.
Fortunately, unlike texts or computer programs, where a single lost bit can render the rest of the data useless, a flipped pixel, or a missing sample in an audio file is hardly noticeable. Lossy compression leverages the fact that Multimedia data can be gracefully degraded in quality by increasingly losing more information. This is what will be explored in this chapter and the next chapters.

Mathematical foundation: Vector Quantization

Consider the following problem: We have an image that we want to store on a certain disk but no matter how hard we try to compress it, it won't fit. In fact, we know that it won't fit because information theory tells us that it cannot be compressed to the size of the space that we have without loosing any information. What choice do we have? Well, we could either not put the image on that disk or we could try to put as much of the image on the disk as we can fit, loosing some of the information.

There are multiple variants of how this could be performed: One could crop the image margins or just extract the parts of the images that are relevant to, let's say a customer. Of course, this would require an expert to be able to judge the relevance of each pixel.

A variant that can be performed automatically in a computer, without knowing anything about the data we are looking at is called vector quantization. Quantization is a general mathematical term and means discretizing a value range using different step sizes. Any given function f can be transformed into a quantized version q by using a transformation g. Often, a real-valued function f is converted to a integer-valued representation. For example:

$$Q(x) = g(\lfloor f(x) \rfloor)$$

Vector quantization involves vectors, i.e. instead of a single-valued function, tuples, triples, or n-dimensional vectors are quantized. In other words, vectors $[x_1,x_2,...,x_k]$ are to be mapped to values $[y_1,y_2,...,yn]$ with $n<<k$. Think of these vectors as being anything, for example 8x8 pixel blocks in an image, red/green/blue triplets in a video, or 16 subsequent samples in an audio file.

So what is the best way to do it? The answer is: There is none but there are many to choose from! Which algorithm to use depends on the type and content of the data and the application. In the following we will present some common approaches.

Linear Quantization

The easiest way to map n vectors to k vectors with k < n is linear quantization. All of the n numbers are distributed evenly into k buckets. This is for example done by simple arithmetic rounding.
The problem with this method is that assuming an even distribution does not often yield not very good results because the distribution of the n numbers is not even. For example, consider an image where colors are to be reduced. Very often there may be a large amount of one color but not such a large amount of a second color.

K-Means
The K-means algorithm is well-known in statistics and machine learning as a clustering algorithm. This is an algorithm to partition n objects into k clusters, where k < n. The algorithm is based on the so-called expectation maximization principle. The idea of the algorithm is to minimize the intra-cluster variance, that is the squared distance between each member of a cluster to the mean value of each member of the cluster should be minimal. Formally, the function V

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

where there are k clusters S_i, i = 1, 2, ..., k, and μi is the centroid or mean point of all the points $x_j \in S_i$ is to be minimized. If the x_j are of higher dimension than 1, a different subtraction function has to be chosen, for example Euclidean distance.

The pseudo-code description of the code looks like this:

```
// Input: a set of data points X, an integer number k
// Output: a set of k data points representing the set μ
k_means(X, k)
    Choose k initial means μ_i from X at random
    WHILE (means μi are not updated significantly anymore)
        // maximization:
        FOREACH (sample x_j: from X)
            assign membership of each element to a mean (closest mean)
        // expectation:
        FOREACH (mean μ_i)
            calculate a new μ_i by averaging x_j that were assigned members
k_means := {μ_i}
```

Variants of the algorithm exist, where the means are not initialized at random but based on some assumptions or properties of the data. In practice, it is often hard to decide when the stopping criterion “not updated significantly anymore” is true. Therefore different criteria are used, often a fixed amount of iteration is preferred (loop n times). K-means is a very popular algorithm for all kinds of quantization tasks and is used in many fields as a “first guess” approach, i.e. try this one before you try anything more complicated. However, K-Means has three major limitations: The computational complexity increases dramatically with large amounts of data and, obviously, one has to know the amount of clusters k in advance. For example, the algorithm is well-suited for reducing 16777216 colors in an image to 65536 colors. But what if we don’t know how many colors are really needed in the image to represent it well? For example, if the image contents is really just black and white but it is still represented by the aforementioned 65536 colors, we will

use more colors than needed and waste memory, network bandwidths, or storage space. However, if we reduce a rainbow image to two colors, we might not be very happy with the appearance of the image in the end.

X-Means

An algorithms that tries to overcome the problem of having to know the number of clusters k in advance by guessing it is the X-Means algorithm. X-Means extends K-Means in several ways, including making the means computation more efficient by effectively caching computation results from previous iterations. The following section will discuss only the heuristics for automatically guessing k. The algorithms starts with a kmin which defines the lower bound of the range where k is to be searched and a kmax which defines the upper bound.

```
// Input: a set of data points X,
// kmin and kmax denoting lower and upper boundaries for k
// Output: a set of k data points representing the set μ
x_means(X)
   k := kmin
      Run k_means(X,k) until it converges
      Score the quality of the clustering
      IF (k>kmax) stop and report the best scoring clustering during the search
x_means := {μi}best
```

The main question with this algorithm is: How do we score the quality of the clustering?
Of course, we cannot do it optimally. If we could do it optimally we could greatly reduce the number of steps in this algorithm in the first place. Also, the optimal clustering depends on the underlying data we are processing and might be different for different types of image, video, and audio data.

However, the authors of the X-Mean algorithm, Pelleg and Moore, proposed the following heuristics that is based on general statistic assumptions. In order to score the quality of a clustering, each mean is split into two children: The children are moved a distance proportional to the size of the region in opposite directions. Next, in each parent region, a local K-Means with k=2 is run for each pair of children. Local means, the children are affected only by the points in the parent's region and not by any other parts of the data. Once this 2-means run has converged a test is performed on all pairs of children. Informally, the test asks: "is there statistical evidence that the two children are modeling a real structure here, or would the original parent model the distribution equally well?".

Many metrics have been defined in statistics to answer this question, none of them works perfectly, since again, the solution to this problem depends on the structure of data one is dealing with and the task one wants to ultimately accomplish. However, an often used metric that seems to work well in many cases is the so called Schwarz Criterion or Bayesian Information Criterion (BIC). In order to use BIC, one has to interpret the means as the mean of a spherical Gaussian [DEFINE IN APPENDIX] that defines a model to describe the data points belonging to the mean. This allows to assign a probability to each of the data points of belonging to the spherical Gaussian. We call, the Gaussian models M, so that M_j is the j-th model (derived from the j-th mean). BIC is then defined as:

$$BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2} \cdot \log R$$

where l^j(D) is the log-likelihood [XXX DEFINE IN APPENDIX] of the data according to the j-th model, and p_j is the number of parameters in M_j (in our case the number p_j is the sum of the k-1 class probabilities, MK mean coordinates, and one variance estimate). R is the total number of data points which belong to the mean under consideration. BIC is a score, which basically favors the model with the minimum number of parameters. In other words: If our newly introduced two means represent the data equally well than one mean, we don't need to introduce two new means.

If BIC determines that the children means describe the data better than the parent, they are chosen instead of the parent (thus increasing k by one). Then the algorithm goes back to step one. Each time, a K-Means runs has converged, a global BIC score is calculated. When k is bigger than k_{max}, the globally computed BIC scores are compared and the best one, i.e. the highest, is chosen.

Perceptual Quantization

The mathematical methods described above are very useful for many tasks, especially if one wants to quantize data that one has no knowledge about other than general statistics. The main limitation of all of the methods described previously in this chapter and most of other methods is that it they tend to converge to local minima. Also, there is no way to tell whether they converged to a local minimum or not. Therefore, when applied to a concrete piece of data, the results can vary greatly, e.g. from one photo to another.

Fortunately, with acoustic and visual data we often have more background knowledge than just the theorems of mathematical statistics. As explained in Chapter XXX [introduction how ear works, etc...], we have a great deal of knowledge of how human perception works. Until the arrival of the digital age, the paradigm was that audio and video content should be reproduced as accurately as possible whenever copied. This means, that when comparing the original and the copy, the difference should be as small as possible. The idea behind perceptual quantization is that even though it is highly desirable to reproduce content as accurately as possible, the term "accurate" is not defined as a simple mathematical distance (such as an L-norm distance or root-mean-square error) but using some perceptual model. In other words: Two signal are accurately reproduced if the sound the same or they look the same even if they have many differences when compared bitwise. In order to achieve this, one needs a model of the perceptual sensors. The following sections describe the most important ideas of perceptual coding.

Sound Amplitude Quantization: A-law and μ-law

One of the simplest and yet most used perceptual quantization techniques for audio data are the A-law and μ-law algorithms. They area also referred to as companding algorithms, which is the older term inherited from analog signal processing. The idea is to exploit Weber-Fechner's law, which attempts to describe the relationship between the physical magnitudes of stimuli and the perceived intensity of the stimuli. As already explained in Chapter XXX, the Weber–Fechner law assumes that just noticeable differences are additive. As a consequence, sound intensity is per-

ceived logarithmically. In other words, a sound must be twice as intense to be perceived a constant factor more intense in the human ear. The idea behind the A-law and μ-law algorithm is to exactly exploit this fact: Rather than quantizing the intensity of the sounds, i.e. the sample values, linearly, they are quantized logarithmically. Thereby mapping slightly different intensities to the same value and loosing small sound intensity changes. Figure 1 shows the idea.

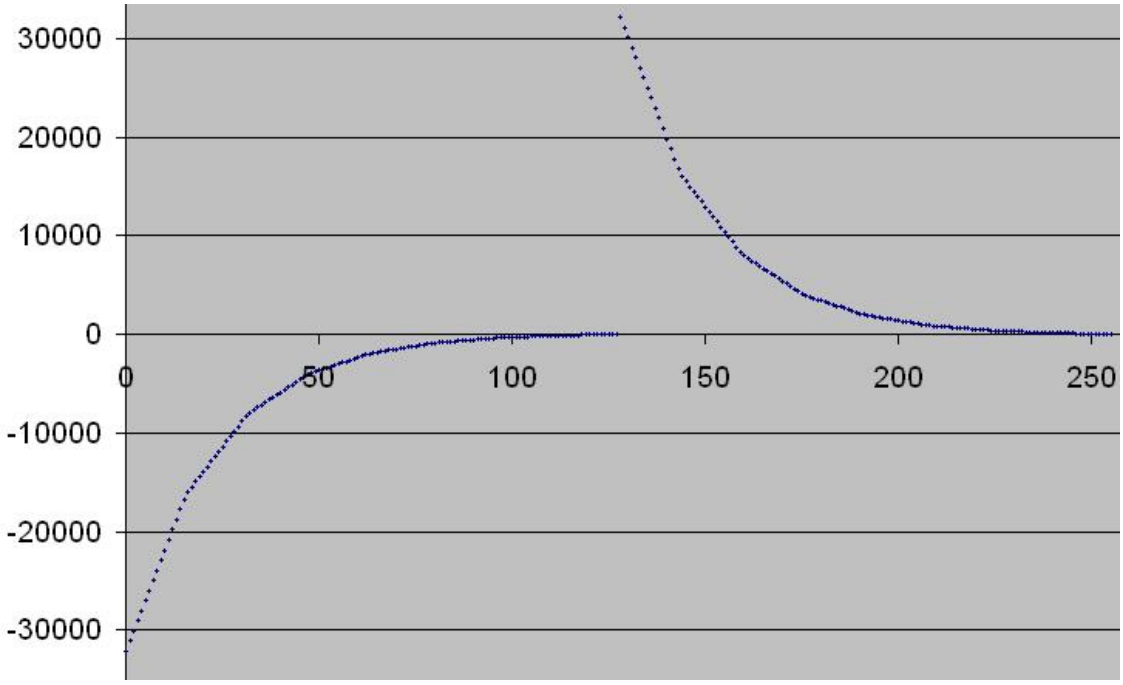

Figure 1. The μ-law quantization. The y-axis shows the input values and the x-axis shows the encoded values. One can see the logarithmic scale and the much denser concentration of values for low-amplitude input signals where the ear is more sensitive.

The companding formula for encoding a sample x normalized to the interval [-1,1] in μ-law is:

$$F(x) = \operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \qquad -1 \le x \le 1$$

where μ is usually 255 for 8 bit and sgn(x) is the sign function. Uncompressing requires the inversion of the formula, which is:

$$F^{-1}(y) = \operatorname{sgn}(y)(1/\mu)[(1+\mu)^{|y|} - 1] \qquad -1 \le y \le 1$$

The companding formula for a-law is:

$$F(x) = \operatorname{sgn}(x)\begin{cases}\frac{A|x|}{1+\ln(A)}, & |x| < \frac{1}{A} \\ \frac{1+\ln(A|x|)}{1+\ln(A)}, & \frac{1}{A} \le |x| \le 1,\end{cases}$$

where A is the compression parameter. In Europe, A = 87.7; the value 87.6 is also used.
Decompression works as follows:

$$F^{-1}(y) = \operatorname{sgn}(y) \begin{cases} \frac{|y|(1+\ln(A))}{A}, & |y| < \frac{1}{1+\ln(A)} \\ \frac{\exp(|y|(1+\ln(A))-1)}{A}, & \frac{1}{1+\ln(A)} \leq |y| < 1. \end{cases}$$

Conceptually, both algorithms are the same. The μ-law algorithm provides a slightly better compression than A-law at the cost of worse proportional distortion for low-amplitude signals. Both of them are used world wide in digital telephony usually for compressing a 16-bit signal into a 12-bit signal. Since A-law is slightly better quality telephony companies agreed that A-law is preferred for an international connection if at least one country uses it. The μ-law algorithm has become a quasi standard for low-bandwidth voice recordings, as it is the default encoding of Sun's audio file format (file extension ".au"). This format is also the default format of Linux' */dev/audio* device and is supported by most common audio API's. It has been standardized as ITU-T Recommendation G.711. Both algorithms are practically implemented by one lookup table for encoding and one lookup table for decoding.

Visual Quantization
Like the ear, different quantities sensed by the eye scale logarithmically with intensity. One example is brightness. This had already been discovered by the ancient Greeks: Stellar Magnitude, which measures the light intensity of stars in the sky and was invited by Hipparchus in about 150 BC has a logarithmic scale. So by using a logarithmic scale for brightness, something like μ-law for audio could be effectively used for image data as well. A system that does this was patented for the first time by A. B. Clark of AT&T in 1928. Many different patents exist in this domain and image data can be quantized in many ways. Therefore there is no one algorithm standard for brightness quantization. We leave the creation of a grayscale image compression algorithm using a brightness compander as an exercise to the reader.

The old NTSC TV standard as well as the JPEG compression algorithm uses quantization in the different color channels. However, this is a different quantization since it involves spatial information.

Motion Quantization
Video itself is the best example for quantization at work. A video is a sequence of images that is replayed in a rapid succession. Just as any other sampling, a video is a quantized version of the reality it represents. The human eye is able to fuse the images into a moving scenes if the images are presented with about 1/30 of a second distance between them. In other words, we need a frame rate of about 30Hz to create the illusion of a moving scene. This, however, is just a rule of thumb. The actual frame rate depends on the physical state of an individual and on the content that is shown. The co-existance of a lip-synchronous audio track, generally allows for lower frame rates because the human brain is good at filling missing information across modalities. Therefore, when wanting to compress a video, it is often adequate to quantize the reality even further and play back a video at about 15 frames per second. A technique very often used for web demonstrations and other bandwidth-critical applications.

Differential Coding

Quantization methods are relatively simple to implement and offer a decent lossy compression scheme. However, they will usually not work anywhere beyond a compression ratio of 2:1 with acceptable perceptual reproduction quality. Differential coding is a scheme that may or may not build on quantization. It leverages global knowledge about the properties of the signal to encode. It is widely used for audio, image, and video compression alike and scales very well from lossless to entirely lossy. The general scheme is presented in Figure 2.

The main component of a differential encoder is the predictor. The predictor takes as input the n samples of the signal to predict the next m samples. The predicted samples are then compared to the original input. The difference is considered to be the encoding. If the prediction was perfect, only zeros would have to be transmitted to the decoder. Of course, nothing is ever perfect but good decoders will produce very small differences that do not need many bits. Different strategies are used to model predictors and the next section will present some of them. To decode the signal, previously decoded samples are used to feed an identical predictor. The prediction is then added to the encoding to reproduce the signal. To bootstrap the process, the first few samples can be predicted with 0 so that the actual signal is transmitted for initialization. This scheme by itself is lossless. However, some predictors have a build in quantization and sometimes the output difference is thresholded. For errors not to become too large over time, some algorithms, alternate between lossy and lossless encoding, eg. every couple of frames, an unencoded sample has to be part of the stream. For better compression, difference encoding is often combined with entropy encoding.

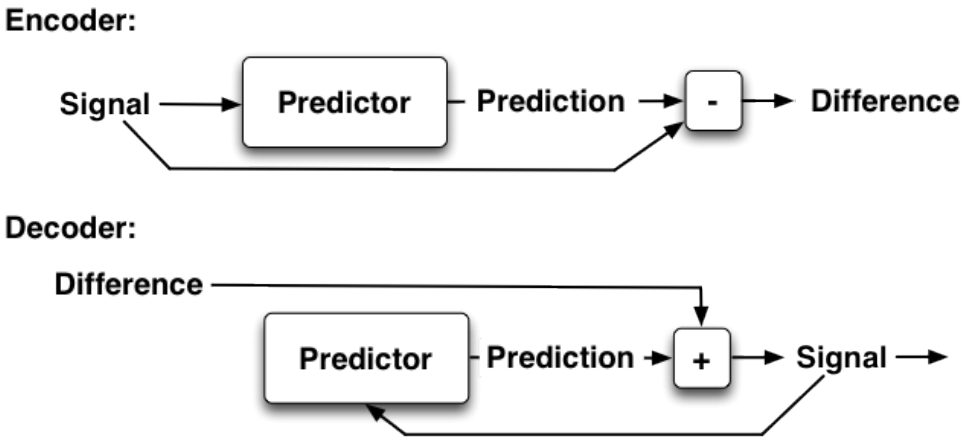

Figure 2. Schematic of a differential encoder: The signal is used as the input for a prediction model. The predictions are then subtracted from the original signal. The difference is transmitted. Decoding uses the previously decoded signals to feed the predictor which must, of course, be identical in both encoder and decoder.

Differential Coding in Audio: ADPCM

As a first example of a differential encoder, we look at ADPCM. ADPCM stands for adaptive differential pulse code modulation and is standardized in ITU-T G.726. It is a wide spread audio format and is used mainly for speech encoding. The idea behind the difference encoding of audio is that sound other than noise follows a rather predictable wave pattern i.e., the differences between consecutive samples will usually be much smaller than the samples themselves. Still, in some cases, sample values might drop from very high amplitude to very low amplitude and sometimes not. To maintain a constant-rate difference bitstream, one must furthermore predict whether these differences are large or not for the next couple of sample values. This, together with a quantization of the difference values is the idea of ADPCM.

The ADPCM algorithm is used to map a series of 12 bit μ-law (or a-law) PCM samples into a series of 4 bit ADPCM samples. Given the original sample values as input, ADPCM predicts both the next sample and the step size. A large step size means that the audio sample differences are large, a small step size means the differences are small. The IMA-ADPCM standard (Interactive Multimedia Association) defines ADPCM as shown in Figure 4. The 4-bit IMA ADPCM code consists of 4-bit: 1 bit for the sign and 3 bits for the difference.

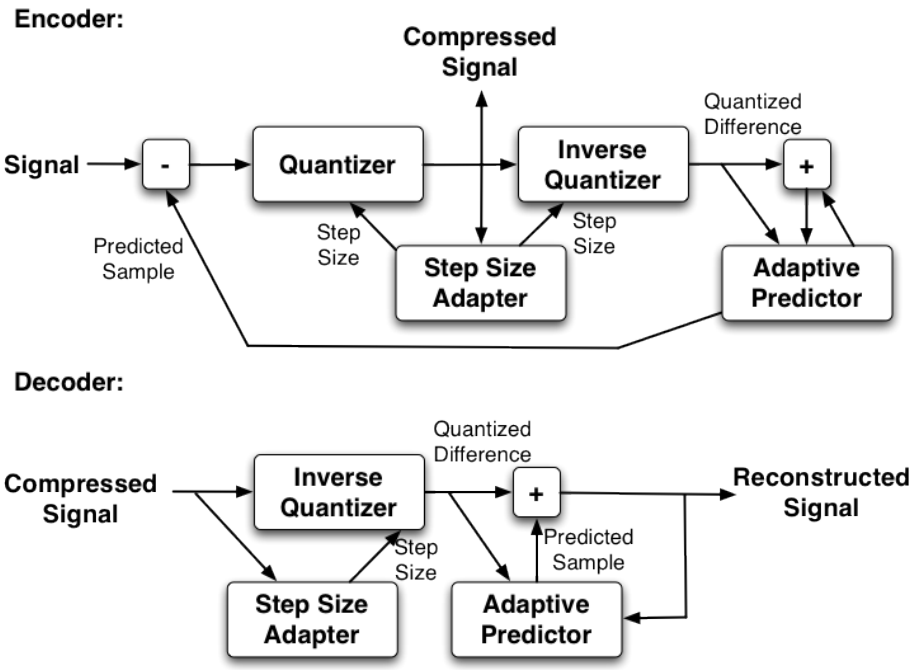

Figure 3. ADPCM encoder and decoder. A differential coder for audio signal that takes into account both the first and the second derivative of the signal. As with any predictive coding algorithms, the decoder block is embedded in the encoder.

Using a prediction model, an adaptive predictor guesses if future samples values might be large. If so, the algorithm increases the bits available for the difference encoding by adjusting the initial quantization. If it is determined that the step sizes might be smaller, the quantization is adjusted to a smaller bit length per sample. For every sample, two values have to be calculated: The difference to the previous (inverse quantized) signal sample and a value that determines the current

step size. The process is bootstrapped by the first values all set to zero and the original sample being passed through.

Decoding works almost the same way: The compressed signal enters the decoder. The encoded value is the difference to the previously predicted value. This difference is added to the previous output value and constitutes the new output value. At the same time, the new step size is calculated which is needed to calculate the next prediction. Since decoder and encoder are very similar and yet ADPCM, we will only provide the pseudo-code for a decoder here. For the concrete values of tables, header infomation, and magic numbers please refer to the standard itself.

```
// Input: a sequence of ADPCM-coded samples C
// Output: a sequence of raw audio samples S
adpcm_decode(C)
    Read header information from C
    First sample of S := first sample in C
    step_size_index := initial index provided in header
    stepsize := stepsizetable[step_size_index]
    oldsample := first sample in C
    FOREACH (sample c in C)
        delta := stepsize encoded in c
        s := oldsample + delta
        Add s to S
        step_size_index := indexTable[s]
        stepsize := stepsizetable[step_size_index]
        oldsample := s
decode_adpcm := S
```

ADPCM achieves a decent compression rate (about 4:1) and is very useable for speech signals. Music and other noise is not compressed very well using this methodology as the quality can suffer badly. It’s not unbearable though so it could be used as a poor man’s music compressor.

Differential coding in Images: PNG

The PNG image format uses a differential encoding step before an LZ-derivate entropy encoder is used. The algorithm predicts the color of each pixel based on the colors of previous neighboring pixels and subtracts the predicted color of the pixel from the actual color. An image line compressed in this way is often more compressible than the raw image line would be, especially if it is similar to the line above, since the differences from prediction will generally be clustered around 0, rather than spread over all possible image values. This is particularly important in relating separate rows, since the later applied entropy compression (DEFLATE see Chapter XXX), has no understanding that an image is a 2D entity, and of course interprets the image data as a stream of bits. The predictor uses the pattern depicted in Figure 3 to sift through the image.

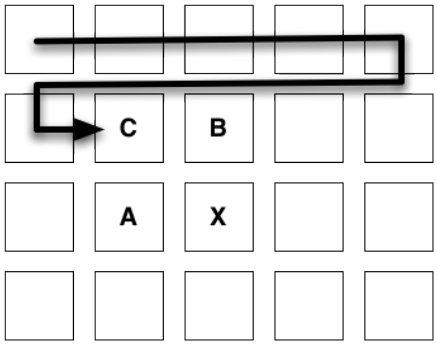

Figure 3. The PNG image is scanned line by line. The color value of pixel x is the one to be predicted and one of five different predictor states can be used that depend on the color values of a, b, and c.

The predictor has five states which predict the value of each byte (of the original image data) based on the corresponding pixel to the left (a), above (b), above and to the left (c) or some combination thereof, and encodes the difference between the predicted value and the actual value. The states are shown in Table 1:

State	Name	Predicted value
0	None	Zero (so that the raw value passes through unaltered)
1	Sub	Value of a (to the left)
2	Up	Value of b (above)
3	Average	Mean of bytes a and b, rounded down
4	Paeth	a, b, or c, whichever is closest to $p = a + b - c$

Table 1. Predictor states used for differential encoding in the PNG image format.

The Paeth filter computes a simple linear function of the three neighboring pixels (a, b, c), then chooses as predictor the neighboring pixel closest to the computed value as defined by the following pseudo-code:

```
// Input: color values a,b, and c as illustrated in Figure 3
// a = left, b = above, c = upper left
// Output: a paeth-prediction for a,b, and c
paeth_predict(a,b,c)
    p := a+b-c
    pa := abs(p-a)
    pb := abs(p-b)
    pc := abs(p-c)
    IF (pa<=pb AND pa<=pc) p := a
    ELSE IF (pb <= pc) p := b
    ELSE p := c
paeth_predict := p
```

Compression of a pixel value x dependent on its neighbors a,b, and c works by calculating

$$compressed(x) = x - paeth\_predict(a,b,c)$$

and decompression works by reversing the formula to

$$uncompressed(x) = compressed(x) + paeth\_predict(a,b,c).$$

The particular states are chosen adaptively on a line-by-line basis based on a heuristic developed by Lee Daniel Crocker, who tested the methods on many images during the creation of the format. If interlacing is used, each stage of the interlacing is filtered separately. It makes the compression generally less effective though.

Differential Coding in Video: Motion Compensation in MPEG-1

Differential encoding is also used for videos. Every video that is shipped on Video CD, DVD, Blue Ray Disk, or through digital cable, satellite, or antenna television is compressed using an algorithm that contains a differential encoder. The idea is that at 25 frames per second or more, the differences between two or even more consecutive video frames are minor. Any object in the camera view that does not change its appearance or position during 0.04 seconds will not have changed in between two frames. Also, physical objects tend to not randomly change position. In other words, an object that is visible on the left and is known to have changed its position rightwards over the last couple of frames will probably either stop in the next frame or continue to do so. This is the idea behind motion compensation: Rather than just encoding video frames as stand-alone images, the difference between them is modeled by a differential encoder. This technique is used in different variations in all versions of MPEG video and in many other video codecs. In the following, we will describe one version of the MPEG-1 motion compensation.

The MPEG-1 algorithm works on a block-by-block basis. In the next chapter we will explain what the advantages of this approach are. For now, it is important to know that an 8x8 pixel block is called macro block. To decrease the amount of spatial redundancy in a video, only macro blocks that change inside a certain amount of consecutive frames are updated. This is known as conditional replenishment. However, conditional replenishment is not very effective by itself. Movement of large objects, and/or the camera may result in large portions of macro blocks needing to be updated, even though only the position of the previously encoded objects has changed but not their appearance. Through motion prediction the encoder can compensate for this movement and remove a large amount of redundant information. The encoder compares the current frame with adjacent parts of the video from the previous frame up to an encoder-specific predefined radius limit from the area of the current macro block. If a match is found, only the direction and distance (i.e. the vector of the motion) from the previous video area to the current macro block need to be encoded.

Of course, a predicted macro block rarely matches the current frame perfectly. The differences between the predicted matching area and the real macro block is therefore the lossy part of the process. The larger the error, the more data must be additionally encoded in the frame.

The distance between two areas in a frame is measured in number of pixels but is often referred to as pels. MPEG-1 video uses a motion vector precision of one half of a pixel or half-pel. The finer the precision, the more accurate the match is likely to be, and the more efficient the compression. Higher precision, however, also requires higher runtime of the encoder and also a larger encoding bitrate since potentially more motion vectors have to be stored. In the end, since neighboring macro blocks are very likely to have similar motion vectors, only the difference between the motion vectors has to be encoded for each macro block. The pseudo-code for a very simple, exhaustive motion estimator is shown below:

```
// Input: REF = reference frame, CUR = current frame
// Output: K a set of indizes to the 8x8 blocks in REF most
// closely the matching the block in CUR
motion_estimate(REF, CUR)
    FOREACH (block MB in CUR)
       FOREACH (i := 0,1,...,64) //8x8 block
           pcur[i] := pixel in MB
           pref[i] := pixel in REFMB
           FOREACH (block k in REF)
               distortion[k] := SUM(distortion(pcur[i],pref[k,i])
       Add minimum value for distortion[k] to K
motion_estimate := K
```

Ideally, the function distortion reflect human perception. Unfortunately, this is still a matter of research, therefore very often Euclidean distance or other similarly simple metrics are used.

In order to bootstrap the process, the first frame is not dependent on any other frame. It is just a JPEG image (so-called I-Frame). The frames after the I-Frames are called P-frames (predictive frames). MPEG also defines B-frames. These frames takes into account the previous and the next frame for motion prediction and can therefore achieve higher compression ratio. However, it is necessary for the player to first decode the next frame sequentially after the B-frame, before the B-frame can be decoded. Therefore B-frames are computationally more complex both in encoding and decoding.

Since motion compensation is lossy and each frame depends on the previous one the error propagates easily and grows with every frame. Therefore, and also for the ability to play a video from a random time position, I-frames are inserted every couple of frames. The I-frame and all frames that ultimately depend on it, are called group of pictures (GOP). A typical GOP size is about 20 frames.

Except ADPCM, differential encoding is rarely used stand alone. Both MPEG and PNG rely on additional compression steps. The next chapter is going to explore the ideas behind them and also other, more advanced, techniques.

Exercises

1.Implement linear quantization and K-means and use the implemented quantization algorithms to reduce the colors of several color images to 2, 4, 8, 16, 64, and 256 colors. Which algorithm performs “best”?

2.Try linear quantization on an audio file with different granularity. What artifacts can you hear?

3.Find or implement a μ-law compression and decompression table. Modify the algorithm such that it compresses down to 4 bits.

4.As described in the text, create a logarithmic compander for light intensity and try it on gray scale images.

5.Try to apply μ-law compression as a color quantization algorithm for an image.

6.Apply an entropy encoder of your choice to a set of companded audio files and compare the compression of applying the same entropy encoding to the uncompanded files. How do you explain the results?

7.Create a simple predictive coder and decoder for a sampled audio file only based only on the difference between two sample values. This algorithm is often referred to as differential pulse code modulation (DPCM).

8.Change your DPCM encoder so that instead of trying to minimize the amount of bits for each sample, it maximizes the amounts of bits (probably even taking more bits than the original). Make sure, it is still a differential compression scheme (although a maximally bad one).

9.Design a differential encoder/decoder for ASCII text files containing natural language. Try to model your predictor so it takes into account the properties of the natural language the original text is encoded in (e.g. the frequency of the individual characters following other characters).

10.Discuss the properties of sound files when ADPCM compression will work well and when it won't. Both compression efficiency and perceptual accuracy should be described.

11.Would the application of a smoothing algorithm help improve the quality of the output of the ADPCM encoder?

12.Implement the PNG difference encoder as described in the text and apply it to some test images. Describe how you choose the heuristics to find the state of the predictor.

13.Add another predictor state to your PNG algorithm -- when would you choose it and why do you think it is good?

14.Extract several consecutive frames from a video and save them as JPEG.

a) Calculate the difference image and discuss why certain pixels are shown in the difference image.

b) Using several frames from the video, implement a routine to calculate the motion vectors for 8x8 blocks.

Literature

- Ken C. Pohlmann (1985). Principles of Digital Audio (2nd ed.). Carmel, Indiana: Sams/Prentice-Hall Computer Publishing. ISBN 0-672-22634-0.
- μ-law: ITU.T Recommendation G.711
- ADPCM: ITU.T Recommendation G.726
-PNG: RFC 2083
-MPEG-1 Video: ISO/IEC-11172-2
-Paeth, A.W., "Image File Compression Made Easy", in Graphics Gems II, James Arvo, editor. Academic Press, San Diego, 1991. ISBN 0-12-064480-0.

Web Links

·Quantization threads in comp.dsp:
http://www.dsprelated.com/comp.dsp/keyword/Quantization.php
·Quantization links in datacompression.info:
http://datacompression.info/Quantization.shtml
·PNG home:
http://www.libpng.org/pub/png/

Research Papers

-Ralph Miller and Bob Badgley: U.S. Patent 3,912,868 filed in 1943: N-ary Pulse Code Modulation.

-Franklin S. Cooper; Ignatius Mattingly (1969). "Computer-controlled PCM system for investigation of dichotic speech perception". Journal of the Acoustical Society of America 46: 115. doi:10.1121/1.1972688.

-P. Cummiskey, N. S. Jayant, and J. L. Flanagan, "Adaptive quantilation in differential PCM coding of speech," Bell Syst. Tech. J., vol. 52, pp. 1105—1118, Sept. 1973.

-J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations", Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1:281-297.

-Dan Pelleg, Andrew Moore: "X-means: Extending K-means with Efficient Estimation of the Number of Clusters", Proceedings of the Seventeenth International Conference on Machine Learning, pp. 727-734, San Francisco, 2000.

-Didier Le Gall: "MPEG: a video compression standard for multimedia applications", Communications of the ACM, vol. 43, no 4, pages 46-56, April 1991.